

## imsai.pas

```

cut_num = 1
for i = 1 to n
  was_cut[i] = false
next
repeat
  for i = 2 to n
    if not (was_cut[i]) then
      cut_it = true
      for j = 2 to n
        if not(was_cut[j]) then
          if _to[j]=i then
            cut_it = false
          end if
        end if
      next j
      if cut_it then
        cut_ord[i] = cut_num
        was_cut[i] = true
        total_cuts = total_cuts + 1
      end if
    end if
  next i
  cut_num = cut_num + 1
until total_cuts = n - 1

procedure prim_tree
passed variables:
n           { number of nodes, including primary SAI }
dmtx        { upper triangle of distance matrix }
*_from     { list of SAIs, with #1 = primary }
*_to       { Bring forward lines from previous microgrids }
*d1         { distance from each SAI to next node }
*d2p        { distance to primary from each SAI }

local constants:
dlarge = 999999999.9

local variables:
i
j
k
l
a
b
c
min
dist
dist2
cost

```

## primsai.pas

```

for i = 1 to n do
  a[i] = true
  b[i] = 0
  c[i] = dlarge
  d1[i] = zero
next
c[1] = zero
d2p[1] = zero
for i = 2 to n
  d2p[i] = dmtx[i][1]
next
j = 1
for i = 2 to n
  min = dlarge
  for k = 2 to n
    if (k <> j) then
      if a[k] then
        dist = dmtx[j][k]
        cost = dist
        if cost < c[k] then
          c[k] = cost
          d1[k] = dist
          b[k] = j
        end if
        if min > c[k] then
          min = c[k]
          l = k
          dist2 = d1[k] + d2p[b[k]]
        end if
      end if
    next k
    j = 1
    a[j] = false
    d2p[l] = dist2
  next i
  for i = 2 to n do
    _from[i] = i
    _to[i] = b[i]
  next
  _from[1] = 1
  _to[1] = 1

procedure get_link_cost
passed variables:
number_of_SAIS

```

imsai.pas

```
primsai.pas

SA
SAI_lines
saix
saiy
density
*link_cost
*term_cost
*link_line_feet
*nc96
*nc24
*ugd_cable
*bur_cable
*aer_cable
*ugd_structure
*bur_structure
*aer_structure
*ManholeCost

local variables:
uc
bc
ac
us
bs
as
mh
m
from
_to
DistToNextTerm
DistToPrimary
cuts
i
j

term_cost = zero
link_cost = zero
ugd_cable = zero
bur_cable = zero
aer_cable = zero
ugd_structure = zero
bur_structure = zero
aer_structure = zero
ManholeCost = zero
link_line_feet = zero

if number_of_SAIs > 1 then
    { First, set up distance matrix between SAIs...}
    for i = 1 to number_of_SAIs
        for j = 1 to number_of_SAIs do
            m[i][j] = abs(saix[i]-saix[j]) + abs(saiy[i]-saiy[j])
        next j
    next i
    call prim_tree
    pass variables:

```

(primsai.pas)

```
n      = number_of_SAIs
dmtx  = m
*_from = _from
*_to   = _to
*d1   = DistToNextTerm
*d2   = DistToPrimary
call prune
pass variables:
n      = number_of_SAIs
_to   = _to
*cut_ord = cuts
call cumulate_lines
pass variables:
n      = number_of_SAIs
_to   = _to
ds0_lines = SAI_lines
DistToNode = DistToNextTerm
DistToPrimary = DistToPrimary
cuts   = cuts
density = density
SA     = SA
*link_cost = link_cost
*term_cost = term_cost
*n96   = nc96
*n24   = nc24
*ugd_cable = uc
*bur_cable = bc
*aer_cable = ac
*ugd_structure = us
*bur_structure = bs
*aer_structure = as
*Manhole_cost = mh
ugd_cable = ugd_cable + uc
bur_cable = bur_cable + bc
aer_cable = aer_cable + ac
ugd_structure = ugd_structure + us
bur_structure = bur_structure + bs
aer_structure = aer_structure + as
ManholeCost = ManholeCost + mh
for i = 1 to number_of_SAIs
    link_line_feet = link_line_feet + SAI_lines[i] * DistToPrimary[i]
next
end if

```

(primsai.pas)

ninal.pas

minal.pas  
ee functions are used outside of this module:

fiber\_terminal\_cost\_fn  
tl\_terminal\_cost\_fn  
drop\_terminal\_cost\_fn

unction fiber\_terminal\_cost\_fn

passed variables:  
lines  
distance  
density  
\*n2016  
\*n672  
\*n96  
\*n24  
pct\_ugd  
pct\_bur  
pct\_aer

local variables

cost

mincost

min2016

min672

min96

min24

i

j

k

12016

1672

196

124

cabcost

uc

bc

ac

uf

bf

af

(Calculates cost of fiber terminals for a given number of DSO lines served,  
including number of terminals of each size, using integer search. )

if lines > half then  
 mincost = 1.0e+16  
 for i = 0 to round( lines / 2016.0 + half )  
 for j = 0 to round( (lines - 2016.0 \* i) / 672 + half )  
 for k = 0 to round( (lines - 2016 \* i - 672 \* j) / 96 + half )  
 n2016 = i  
 n672 = j  
 n96 = k  
 n24 = round((lines - 2016 \* n2016 - 672 \* n672 - 96 \* n96)/24 + half )  
 if n24 < 0 then n24 = 0

terminal.pas

12016 = min(2016.0 \* n2016, lines )  
1672 = min( 672.0 \* n672, lines - 12016 )  
196 = min( 96.0 \* n96, lines - 12016 - 1672 )  
124 = lines - 12016 - 1672 - 196  
cost = a2016 \* n2016 + b2016 \* 12016 + a672 \* n672  
 + b672 \* 1672 + a96 \* n96 + b96 \* 196 + a24 \* n24 + b24 \* 124  
  
cabcost = call feed\_cable\_cost  
pass variables:  
lines = (n2016 + n672 + n96 + n24) \* 4 / FiberFillFactor  
density = density  
technology = fiber  
\*ugd\_copper = uc  
\*bur\_copper = bc  
\*aer\_copper = ac  
\*ugd\_fiber = uf  
\*bur\_fiber = bf  
\*aer\_fiber = af  
pct\_ugd = pct\_ugd  
pct\_bur = pct\_bur  
pct\_aer = pct\_aer  
  
cost = cost + cabcost \* distance  
  
if cost < mincost then  
 mincost = cost  
 min2016 = n2016  
 min672 = n672  
 min96 = n96  
 min24 = n24  
end if  
  
next k  
next j  
next i  
  
n2016 = min2016  
n672 = min672  
n96 = min96  
n24 = min24  
  
cabcost = call feed\_cable\_cost  
pass variables:  
lines = (n2016 + n672 + n96 + n24) \* 4 / FiberFillFactor  
density = density  
technology = fiber  
\*ugd\_copper = uc  
\*bur\_copper = bc  
\*aer\_copper = ac  
\*ugd\_fiber = uf  
\*bur\_fiber = bf  
\*aer\_fiber = af  
pct\_ugd = pct\_ugd  
pct\_bur = pct\_bur  
pct\_aer = pct\_aer

minal.pas

```
fiber_terminal_cost_fn = mincost - cabcost * distance
else
  n2016 = 0
  n672 = 0
  n96 = 0
  n24 = 0
  fiber_terminal_cost_fn = zero
end if

action t1_terminal_cost_fn
  passed variables:
  lines
  *nc96
  *nc24

  local variables:
  cost
  mincost
  min96
  min24
  i
  196
  124

  !Calculates cost of t-1 terminals for a given number of DSC lines served, including
  !number of terminals of each size, using integer search. !

  if lines > half then
    mincost = 1.0e+16
    for i = 0 to round(lines / 96 + half)
      nc96 = i
      nc24 = round((lines - 96 * nc96) / 24 + half)
      if nc24 < 0 then nc24 = 0
      196 = min(96 * nc96, lines)
      124 = lines - 196
      cost = ac96 * nc96 + bc96 * 196 + ac24 * nc24 + bc24 * 124
      if cost < mincost then
        mincost = cost
        min96 = nc96
        min24 = nc24
      end if
    next
    nc96 = min96
    nc24 = min24
    t1_terminal_cost_fn = mincost
  else
```

terminal.pas

```
  nc96 = 0
  nc24 = 0
  t1_terminal_cost_fn = zero
end if

function drop_terminal_cost_fn
  passed variables:
  lines
  density
  pct_ugd
  pct_bur
  pct_aer

  local variables:
  i
  temp

  temp = zero
  if lines < 1.0e-6 then
    drop_terminal_cost_fn = zero
  else
    temp = zero
    for i = 1 to NumDropTerminalSizes
      if lines >= DropTermCost[i].size then
        temp = pct_ugd * DropTermCost[i].CostUgd
          + pct_bur * DropTermCost[i].CostBur
          + pct_aer * DropTermCost[i].CostAer
      end if
    next
    drop_terminal_cost_fn = temp
  end if
```

a.pas

```
:h.pas
: only procedure used outside of this modules is calculate_feeder_technology

procedure calculate_feeder_technology
  passed variables:
    feeder_distance
    1
    density
    FillFactor
    *technology
    *n2016
    *n672
    *n96
    *n24
    pct_ugd
    pct_bur
    pct_aer

  local variables:
    n
    tmp1
    tmp2
    tmp3
    c26
    c24
    ct1
    cf
    126
    124
    lt1
    if
    uc
    bc
    ac
    uf
    bf
    af

    n2016 = 0
    n672 = 0
    n96 = 0
    n24 = 0
    technology = copper26

    SA_array[i].fiber_terminal_cost = zero
    SA_array[i].t1_terminal_cost = zero
    SA_array[i].interface_cost = zero

    SA_array[i].n2016 = 0
    SA_array[i].n672 = 0
    SA_array[i].n96 = 0
    SA_array[i].n24 = 0
    SA_array[i].nc96 = 0
```

tech.pas

```
SA_array[i].nc24 = 0
126 = SA_array[i].ResLines / FillFactor
  +(SA_array[i].BusLines - 11 / 12 * SA_array[i].SwitchedDS1
  - 11 / 12 * SA_array[i].SpclAccessDS1) / FillFactor

124 = 126
lt1 = (SA_array[i].ResLines / FillFactor + SA_array[i].BusLines / FillFactor)
  * t1_redundancy_factor / 12
  (terminal.pas)

tmp1 = call fiber_terminal_cost_fn
  pass variables:
    lines = SA_array[i].lines / FillFactor
    distance = feeder_distance
    density = SA_array[i].density
    *n2016 = n2016
    *n672 = n672
    *n96 = n96
    *n24 = n24
    pct_ugd = pct_ugd
    pct_bur = pct_bur
    pct_aer = pct_aer

tmp1 = tmp1 * ac_fib_term
if = (n2016 + n672 + n96 + n24) * 4 / FiberFillFactor
  ( Calculate provisional terminal costs. Note that the terminal cost fns use DSO
  equivalent lines, so we need the fill factor, but not DSI calculations. )

tmp2 = call t1_terminal_cost_fn
  pass variables:
    lines = SA_array[i].lines / FillFactor
    *nc96 = n96
    *nc24 = n24
  (terminal.pas)

tmp2 = tmp2 * ac_t1_term

tmp3 = zero
for n = 1 to NumXCBBoxSizes
  if 126 >= IntfcCost[n].NumLines then
    tmp3 = IntfcCost[n].cost
  end if
next
tmp3 = tmp3*ac_fdi

  ( We will choose feeder technology by least-cost under the assumption that each
  SA sends feeder directly to the switch without sharing cable. )

c26 = call feed_cable_cost
  pass variables:
    lines = 126
    density = density
  (cable.pas)
```

ch.pas

```
technology = copper26
*ugd_copper = uc
*bur_copper = bc
*aer_copper = ac
*ugd_fiber = uf
*bur_fiber = bf
*aer_fiber = af
pct_ugd = pct_ugd
pct_bur = pct_bur
pct_aer = pct_aer

c26 = (uc * ac_ugd_cop + bc * ac_bur_cop + ac * ac_ser_cop
+ uf * ac_ugd_fib + bf * ac_bur_fib + af * ac_ser_fib)
* feeder_distance + tmp3

c24 = call feed_cable_cost
pass variables:
lines = 124
density = density
technology = copper24
*ugd_copper = uc
*bur_copper = bc
*aer_copper = ac
*ugd_fiber = uf
*bur_fiber = bf
*aer_fiber = af
pct_ugd = pct_ugd
pct_bur = pct_bur
pct_aer = pct_aer

c24 = (uc * ac_ugd_cop + bc * ac_bur_cop + ac * ac_ser_cop
+ uf * ac_ugd_fib + bf * ac_bur_fib + af * ac_ser_fib)
* feeder_distance + tmp3

ctl = call feed_cable_cost
pass variables:
lines = ltl
density = density
technology = t_1
*ugd_copper = uc
*bur_copper = bc
*aer_copper = ac
*ugd_fiber = uf
*bur_fiber = bf
*aer_fiber = af
pct_ugd = pct_ugd
pct_bur = pct_bur
pct_aer = pct_aer

ctl = (uc * ac_ugd_cop + bc * ac_bur_cop + ac * ac_ser_cop
+ uf * ac_ugd_fib + bf * ac_bur_fib + af * ac_ser_fib)
* feeder_distance + tmp2

cf = call feed_cable_cost
pass variables:
lines = lf
```

(cable.pas)

)

(cable.pas)

(cable.pas)

tech.pas

```
density = density
technology = fiber
*ugd_copper = uc
*bur_copper = bc
*aer_copper = ac
*ugd_fiber = uf
*bur_fiber = bf
*aer_fiber = af
pct_ugd = pct_ugd
pct_bur = pct_bur
pct_aer = pct_aer

cf = (uc * ac_ugd_cop + bc * ac_bur_cop + ac * ac_ser_cop
+ uf * ac_ugd_fib + bf * ac_bur_fib + af * ac_ser_fib)
* feeder_distance + tmp1

technology = copper26

if (c24 < c26)
or (feeder_distance + SA_array[i].MaxDistance > copper_gauge_xover) then
technology = copper24
end if

if ((ctl < min(c24, c26))
or (feeder_distance + SA_array[i].MaxDistance > max_copper_distance)
or (feeder_distance > copper_t1_xover)) then
technology = t_1
end if

if (cf < min(min(c24, c26), ctl))
or (feeder_distance > tl_fiber_xover) then
technology = fiber
end if

SA_array[i].feeder_technology = technology

if technology = fiber then
SA_array[i].fiber_terminal_cost = call fiber_terminal_cost_fn (terminal.pas)
pass variables:
lines = SA_array[i].lines / FillFactor
distance = feeder_distance
density = SA_array[i].density
*n2016 = n2016
*n672 = n672
*n96 = n96
*n24 = n24
pct_ugd = pct_ugd
pct_bur = pct_bur
pct_aer = pct_aer

SA_array[i].n2016 = n2016
SA_array[i].n672 = n672
SA_array[i].n96 = n96
SA_array[i].n24 = n24
```

## tech.pas

```
elseif technology = t_1 then
  SA_array[i].tl_terminal_cost = call t1_terminal_cost_fn      (terminal.pas)
    pass variables:
    lines = SA_array[i].lines / FillFactor
    *nc96 = n96
    *nc24 = n24

SA_array[i].nc96 = n96
SA_array[i].nc24 = n24
n2016 = 0
n672 = 0

else
  { technology is analog }

  SA_array[i].interface_cost = tmp3

  { Add in switched DS1 line terminals }

if (technology = copper26) or (technology = copper24) then
  SA_array[i].tl_terminal_cost = SA_array[i].tl_terminal_cost
  + call t1_terminal_cost_fn  (terinal.pas)
  pass variables:
  lines = (SA_array[i].SwitchedDS1
            + SA_array[i].SpclAccessDS1) * 12
  *nc96 = n96
  *nc24 = n24

  SA_array[i].nc96 = SA_array[i].nc96 + n96
  SA_array[i].nc24 = SA_array[i].nc24 + n24
  n2016 = 0
  n672 = 0
  n96 = SA_array[i].nc96
  n24 = SA_array[i].nc24
end if

end if
```

## lotdiv.pas

```
lotdiv.pas
the only procedure used outside this module is lot_divide

procedure lot_divide
  passed variables:
  number_of_lots
  var NS_lots
  var EW_lots

  local constatns:
  two = 2

  local variables:
  i
  waste
  minwaste
  NS_try
  EW_try
  NS_try_d
  sqnl
  sqrt2

  { This procedure minimizes wasted lots within a square microgrid, subject
  to the constraint that lots have lengths no more than twice their widths.
  It returns the "optimal" number of lots in the NS and EW direction. }

  NS_lots = one
  EW_lots = one

  if (number_of_lots > one) then
    sqrt2 = sqrt(two)
    waste = number_of_lots
    minwaste = number_of_lots
    sqnl = sqrt(number_of_lots)
    for i = round(sqnl / sqrt2) to round(sqnl) + 1

    { Check from square root of number of lots/2 to square
    root of number of lots. This guarantees that max
    length - width ratio is no more than 2. }

    EW_try = i
    NS_try_d = number_of_lots / EW_try
    NS_try = round(NS_try_d)
    waste = NS_try * EW_try - number_of_lots
    if (waste < 0) then waste = number_of_lots
    if (waste <= minwaste) then
      minwaste = waste
      EW_lots = round(EW_try)
      NS_lots = round(NS_try)
    end if
  next

  else
```

lotdiv.pas

```
EW_lots = round(1)
NS_lots = round(1)
end if
```

**global.pas – data structures**

```
global.pas

Data Structures

type d4vector = array[1..4] of double
i4vector = array[1..4] of integer

d50 = array[1..50] of double
d50_ptr = ^d50

d4vector_ptr = ^d4vector
i4vector_ptr = ^i4vector

string8 = string[8]
string4 = string[4]
string12 = string[12]

techtype = (copper26, copper24, t_1, fiber)

d50x50 = array[1..50,1..50] of double
i50x50 = array[1..50,1..50] of integer

d50x50_ptr = ^d50x50
i50x50_ptr = ^i50x50

type GridRecordType = record
  nrow, ncol : integer
  MicroGridNS, MicroGridEW : double
  cx1, cyl,
  cx2,cy2,
  cx3,cy3,
  cx4,cy4 : d4vector
  qTotalLines, qHouseholds, qBusinessLines, qPrivateLines, qSpecialLines : integer
  LowerLeftX, LowerLeftY, UpperRightX, UpperRightY : double
  Households, Buslines : array[1..50,1..50] of integer
  SwitchX, SwitchY : double
  quadrant : integer
  DepthToBedrock : double
  Hardness : string4
  SoilTexture : string8
  WaterTb : double
  MinSlope : double
  MaxSlope : double
  CBG : string12
end
CoordinateRecordType = record
  OriginX, OriginY, reference_latitude : double
end

t1 = record
  CableSize : integer
  CostUgd : double
  CostBur : double
  CostAer : double
end
t1_ptr = ^t1
```

**global.pas – data structures**

```
t2 = record
  size : integer
  CostBur : double
  CostAer : double
  CostUgd : double
end
t2_ptr = ^t2

t3 = record
  size : integer
  CostUgd : double
  CostBur : double
  CostAer : double
end
t3_ptr = ^t3

t4 = t3
t4_ptr = ^t4

t7 = record
  NumLines : integer
  Cost : double
end
t7_ptr = ^t7

t8 = record
  density : double
  FeedUgd : double
  DistUgd : double
  FeedBur : double
  DistBur : double
  FeedAer : double
  DistAer : double
end
t8_ptr = ^t8

t9=t8
t9_ptr = ^t9

t10 = t8
t10_ptr = ^t10

t11 = record
  DuctCap : integer
  NormalCost : double
  SoftCost : double
  HardCost : double
end
t11_ptr = ^t11

t12 = record
  density : double
  ManholeSpacing : double
end
t12_ptr = ^t12

t13 = record
```

**global.pas – data structures**

```
  density : double
  UgdPct : double
  BurPct : double
  AerPct : double
end
t13_ptr = ^t13

t14=t13
t14_ptr = ^t14

t15=t13
t15_ptr = ^t15

t16 = record
  density : double
  FeedFillFactor : double
  DistFillFactor : double
end
t16_ptr=^t16

t22 = record
  texture : string8
  impact : integer
end
t22_ptr=^t22

t23=record
  density : double
  bur_share : double
  ugd_share : double
  aer_share : double
end
t23_ptr=^t23

IntfcType = (primary,secondary)

GaugeType = (g19, g22, g24, g26)

SARecordType = record
  number_of_SAIs : integer
  TypeOfSAI : array[1..4] of IntfcType
  x : d4vector
  y : d4vector
  SAI_lines : d4vector
  SwitchX : double
  SwitchY : double
  households : double
  lines : double
  ResLines : double
  BusLines : double
  SpclAccessLines : double
  SpclAccessDSL : double
  SwitchedDSL : double
  Grid_Distribution_Cost : double
  MaxDistance : double
  n2016, n672, n96, n24 : integer
  nc96, nc24 : integer
end
```

lobal.pas – data structures

```

snc96,snc24      : integer
fiber_terminal_cost : double
t1_terminal_cost   : double
secondary_tterm_cost : double
interface_cost    : double
drop_cost         : double
nid_cost          : double
drop_terminal_cost : double
feeder_technology : techtype
grid_line_feet    : double
link_line_feet    : double
grid_drop_feet    : double
density           : double
DepthToBedrock    : double
Hardness          : string4
SoilTexture        : string8
WaterTb            : double
MinSlope           : double
MaxSlope           : double
DistToSwitch       : double
subfeeder_try      : double
subfeeder_coord    : double
quadrant          : integer
ugd_cable          : double
bur_cable          : double
aer_cable          : double
ugd_structure      : double
bur_structure      : double
aer_structure      : double
ManholeCost        : double
lines_served       : double
CBG                : string12
end

SARecordType_ptr = ^SARecordType
GridRecordType_ptr = ^GridRecordType

glsarray = ARRAY [1..8000] OF double
glrarray = ARRAY [1..8000] of SARecordType_ptr

glsarray_ptr = ^glsarray
glrarray_ptr = ^glrarray

type d8000 = array[1..8000] of double
i8000 = array[1..8000] of integer
b8000 = array[1..8000] of boolean

d8000_ptr = ^d8000
i8000_ptr = ^i8000
b8000_ptr = ^b8000

d8000x8000 = array[1..8000] of d8000_ptr
d8000x8000_ptr = ^d8000x8000
type d8000 = array[1..8000] of double;
i8000 = array[1..8000] of integer;
b8000 = array[1..8000] of boolean;

```

global.pas – data structures

```

d8000_ptr = ^d8000;
i8000_ptr = ^i8000;
b8000_ptr = ^b8000;

d8000x8000 = array[1..8000] of d8000_ptr;
d8000x8000_ptr = ^d8000x8000;

type d4 = array[1..4] of double;
i4 = array[1..4] of integer;
b4 = array[1..4] of boolean;

d4_ptr = ^d4;
i4_ptr = ^i4;
b4_ptr = ^b4;

d4x4 = array[1..4] of d4_ptr;
d4x4_ptr = ^d4x4;

```